ACCURATE PACKET TIMESTAMPING ON LINUX PLATFORMS FOR PRECISE CAPACITY ESTIMATION

An Implementation of a Highly Accurate Timestamping System Embedded in the Linux Kernel and its Application to Capacity Estimation

David Montoro-Mouzo, Josemaria Malgosa-Sahanuja,
Pedro Piñero-Escuer, Juan Pedro Muñoz-Gea and Pilar Manzanares-Lopez
Dept. of Information Technologies and Communications, Polytechnic University of Cartagena, Cartagena, Spain

Keywords: Capacity Measurement, Packet-Dispersion Techniques, Timestamping, Linux Kernel Networking.

Abstract: In this work, a tool based on packet-dispersion techniques for remotely measuring link capacities is presented. For implementing these techniques, a highly accurate packet timestamping system is presented. This system is fully integrated in the Linux kernel which makes possible to measure extremely exact packet-arrival times. The logic of the measurement system is integrated into a GUI, which reduces the comlexity of using the tool. Finally, the results of a measurement experiment for testing the developed tool are shown, and the lines of future work are exposed.

1 INTRODUCTION

Although a lot of research has been done in the field of capacity measurement, a reliable and versatile tool for remotely measuring high capacities is still needed. The recent increase in the links capacities and its implications in the new designs of the networkings systems origins the necesity of performing an analysis on the current situation of the packet dispersion techniques. The mayor contribution of this paper is analysing the main implications of the modern networking and communication systems to the packet dispersion techniques. As result of these solutions, a new tool for remote capacity measuring have been developed.

Measuring the interarrival-times in the packets very accurately is the main achievement that is necessary to fulfill for a successful implementation of the packet-dispersion techniques. For links with capacity values in the order of gigabit per second, using this techniques implies measuring interarrival-times in the order of hundred of microseconds (in the best case) or in the order of a few hundred of picoseconds (in the worst case). For this reason, the developments proposed in this paper include a Linux based accurate timestamping system.

2 PACKET-DISPERSION

The basics of the packet-dispersion techniques were originally proposed in (Jacobson and Karels, 1988), where it is demonstrated that it is possible to control the congestion on one link using the interarrival-times of the packets sent through it. A good revision of these techniques can be seen in (Prasad et al., 2003). The capacity measurement tool that makes use of the developed timestamp system is based in packet-dispersion techniques. The techniques used in this work are a simplification of the ones proposed in (Harfoush et al., 2003). Basically, in this paper it is proposed an iterative method for estimating remotely the capacity of one link in the path of two hosts. It defines three kinds of individual measures and a sequential process for obtaining the intended link-capacity estimation by using them.

Each one of thesef three individual measures is designed for estimating the capacity of one kind of path. In this manner, the *packet-pair* technique is used to estimate the capacity of a complete path. The *padding-packets* technique estimates the capacity of a prefix of the path (i.e., the capacity of the slowest link of a sub-path at the beginning of the whole path). Finally, the *variable packet-sizes* technique estimates the capacity of a path suffix (i.e., the capacity of the slowest link of a sub-path at the end of the whole

Montoro-Mouzo D., Malgosa-sanahuja J., Piñero-Escuer P., Muñoz-Gea J. and Manzanares-Lopez P..
ACCURATE PACKET TIMESTAMPING ON LINUX PLATFORMS FOR PRECISE CAPACITY ESTIMATION - An Implementation of a Highly Accurate Timestamping System Embedded in the Linux Kernel and its Application to Capacity Estimation.
DOI: 10.5220/0003478701370142
In *Proceedings of the 6th International Conference on Software and Database Technologies* (ICSOFT-2011), pages 137-142
ISBN: 978-989-8425-76-8
Copyright © 2011 SCITEPRESS (Science and Technology Publications, Lda.)
137

path). It is possible to estimate the capacity of any link in a path by combining this three techniques.

2.1 Measurement Methodology using the Three Individual Measures

With the three individual measurement techniques it is possible to know the smallest capacity of an end-to-end path, or each prefix capacity or each suffix capacity. Therefore, it is possible to implement a procedure for measuring every single link-capacity in a path.

The general process used in this work for measuring the capacity of the $i-th$ link of a path (namely c_i) is as follows:

1. Use the packet-pair technique for measuring the capacity of the whole path (i.e., the bottleneck link). This value will be used for dimensioning the padding-packets measurement in the next step (i.e., determining the required number of padding-packets).
2. The second step includes performing two measures. The first one (namely $c_{0,i-1}$) is the measure of the capacity of the prefix until link *i-1* (i.e. just one link before the link under measurement). The second one (namely $c_{0,i}$) is the measure of the capacity of the prefix including the link under measurement. Both measures are obtained using the padding-packets technique, determining the number of padding-packets from the measure of point one. At this time, taking into account that with this two measures the smallest capacity of each one of the two prefixes is known, there are two possibilities:
   · If $c_{0,i-1} > c_{0,i}$ then the estimation of the link-capacity is $c_i = c_{0,i}$ and the measurement process ends here.
   · If $c_{0,i-1} \leqslant c_{0,i}$ then proceed to point three.
3. Since it is not possible to measure the capacity using packet-pairs and padding-packets, the capacity is estimated using the variable packet-size technique. This is done in the last place because this technique is more inaccurate than the other two.

However, the measure of the capacity of the first link of a path (namely c_1) constitutes a special case. Here the procedure above is not applicable, and it is necessary to perform the next steps:

1. Use the packet-pair technique for measuring the capacity of the whole path. This value will be used for dimensioning the padding-packets measurement in the next step.
2. Using the padding-packets technique, measure the capacity of the prefix from the beginning of the path composed of the first link (namely $c_{0,1}$). In this manner, the final estimation of the link-capacity is $c_1 = c_{0,1}$.

3 ACCURATE TIMESTAMPING

The manner in which Linux solves networking questions determines both the implementation and the accuracy of the capacity estimation tool.

3.1 Kernel Space and User Space

First, it is necessary to consider that in Linux the system memory is divided into two distinct regions: kernel space and user space (Bovet and Cesati, 2005).

Two of the most important services provided by the kernel are networking functions and accessing to hardware devices. These services are provided to the user space in the form of system calls. However, using a system call implies data copy between the two spaces. This is called context switching and implies a variable time delay.

3.2 Packet Reception in Linux Kernel

A summary of the whole packet reception process can be found in (Benvenuti, 2005). Here the most relevant questions relative to accurate timestamping packets will be shown.

3.2.1 Top Halves and Bottom-halves

Linux kernel packet reception facility is based in the *bottom-half/top-half* approximation (Love, 2004); which solves the question of receiving packets from the NIC driver at high rate without losses.

The top-half is executed as soon as an interrupt is fired for performing the most critical tasks. This is done in interruption context and with interrupts disabled. In the case of packet reception, the top-half is implemented in two parts: one part written in the device driver (the specific interrupt handler), and another part written in the file */net/core/dev.c* (the general functions for introducing packets in the kernel).

On the other hand, the bottom-half is scheduled by the top-half for delaying the rest of the packet processing to run at a more convenient time. Since the bottom-half executes with interrupts enabled, it can be preempted (Love, 2004) by the top-half. By doing this, no packet loss due to the intensive part of the packet processing is guaranteed. In the case of packet reception, the bottom-half is implemented using a technique called *soft_irqs* (Love, 2004).

3.2.2 NAPI and NOT NAPI Top-halves

The top-half part can be implemented in two styles: the *classic style* and the *NAPI* (New API) *style* (Benvenuti, 2005). This has deep implications for accurate timestamping arriving packets.

In the classic implementation, the arriving packets are processed by the top-half one by one; therefore, every time a packet is received, the interrupt context is accessed. The main kernel function implementing the classic top-half is the `netif_rx()` function that is invoked every time an interrupt is fired. The problem with this implementation is that in high speed interfaces, under heavy traffic conditions, entering the interrupt context for every packet could imply that the packets are processed slower than received. This can cause NIC buffer overload and packet losing.

For solving these issues, NAPI was introduced in the 2.4.20 kernel version. It implements the top-half with a combination of polling and interrupts for minimizing the time spent in interrupt context. The main functions implementing the NAPI top-half in the kernel are `netif_rx_schedule()` and `napi_schedule()`.

It is not in the scope of this paper deeply explain NAPI, but it is important to note that in this context, the real arrival-time of the packets is lost. According to this, it is necessary to use the classic top-half style to obtain precise and real packet arrival-times.

4 IMPLEMENTATION

4.1 General Scheme

An schematic representation of the developed tool can be seen in Figure 1, showing its different blocks.

The logic of the measurement system is integrated into a Graphical User Interface (*Logic Module* in the scheme). This logic module performs the measuring procedure using the services of an external user-space application (*Support Application* in the scheme). This application acts as a middleware between the logic module and the kernel part of the tool, making the execution of the individual measures possible. In this manner, the real control of the kernel functionality is done by this application.

The timestamping and packet sending functionalities are implemented in the kernel, which are the core of the tool. For this, a new socket family was developed from the Linux Raw Socket family (module labeled as *Modified Raw Socket* in Figure 1). This new kind of socket sends and receives the prove packets with no intervention from the user space (i.e., without context switching), in order to fulfill the back-to-back traffic generation requirement.

The highly accurate packet timestamping system is composed of both the socket family and the Linux kernel top-half functions (the modified version). In the top-half reception functions, the arrival-times of the packets of the specific protocol are measured and introduced in the data field of the packets. By doing this, the goal of taking accurate arrival-times is fulfilled. These times are later extracted from the packets at socket level, making possible a high accurate capacity estimation.

Furthermore, as it has been explained in previous sections, the drivers of the NICs were also modified for obtaining real packet arrival-times.

4.2 Logic Module and Support App

As it has been said, the term *Logic Module* refers to an user application developed for controlling the measurement process and retrieving the results. It is written in Java and includes a graphical user interface written using SWT (Standard Widget Toolkit).

Using this interface, the user of the tool defines the remote host for performing the capacity measure (which, of course, has also to be using the tool). After this, the Logic Module performs a traceroute for obtaining the different hopes in the path between local host and remote host, and displays the obtained path in the interface. Then, the user selects the link which capacity is intended to be measured and indicates the Logic Module starting the measure procedure. At this moment, the Logic Module coordinates the execution of the different individual measures (via the Support Application) and stores all the individual results. Once all this process is done, the Logic Module computes the final result and shows it in the interface.

The Support Application is written in C language and provides services to the Logic Module, connecting it with the in-kernel implementation. In this manner, it sends the requests from the Logic Module to the Modified Socket using a set of *IOCTL calls* (Benvenuti, 2005) written for that purpose.

4.3 Modified RAW Socket

It has been developed using the Linux Raw Socket Family (written in the */net/packet/af_packet.c* file), adding to this kind of sockets capabilities for performing the individual measures with no intervention from user space. Like the rest of the Linux kernel, it is written in C.

It performs the individual measures by coordinating itself with its homologue in the remote host. For doing this, this module has two working modes: one when it acts as the host who initiates the measure and send the packets (Sending Mode), and other when it acts as the host who receives the packets and collects the times (Receiving Mode).

On the one hand, when working in Sending Mode it injects back-to-back traffic in the path and waits for receiving the interarrival-time results from the socket in the other side of the path. The `get_cpu()` function is used before packet sending with the objective of avoiding that the Linux Scheduler could preempt or balance our task (Bovet and Cesati, 2005). By doing this, it is ensured that packets are sent back-to-back.

On the other hand, when the Modified RAW Socket works in Receiving Mode, it reads the arrival-times from the packets and stores them. When all the packets have arrived, it sends all the collected times to the host who injected the traffic using the Socket communication.

4.4 Modified Networking Kernel and NICs Drivers

As it has been said, the top-half code has been modified for making possible highly accurate arrival-times measurement. Since the drivers were also modified for not using NAPI, the packets timestamping is done in the `netif_rx()` function of the */net/core/dev.c* file. With this approximation the timestamping system obtains the more precise times possible at kernel level.

For measuring very little times like the ones that it is intended to measure, it is necessary a high resolution clock. For solving this question, there is a variety of possibilities (Bovet and Cesati, 2005) on Linux environments. After testing most of them, using the *Time Stamp Counter* (TSC) was chosen. The TSC is a 64-bit register present in most nowadays Intel machines which is increased by one for every clock cycle. It starts from zero after every reboot.

In our implementation, the Intel NICs models were using the *82566 Gigabit Ethernet* and *82559 Fast Ethernet* controllers. These models use the *e1000e* and the *e100* Linux drivers respectively. It was necessary to modify the current drivers for disabling NAPI receiving scheme. After modifying both drivers, the packets are now introduced one-by-one in the kernel flow of processing via the *netif_rx()* function.

The next question to solve was the interrupt moderation mechanism. In this specific case, it is necessary to disable the interruption moderation procedure for the two Intel's NIC models used. The *e1000e* driver offers the possibility of disabling this procedure at the time that the Linux module is loaded, but the *e100* driver requires to tune some parameters in its code and to recompile it.

4.5 Measurement Procedure

As shown in Figure 1, the procedure for performing a capacity measure between two hosts can be explained like a sequence of consecutive steps. In this figure, the host which wants to measure the capacity (i.e., the host which sends the packets) is labeled as *Local host*, whereas the host receiving the packets (i.e., the host measuring the times) is labeled as *Remote host*. In this manner, the sequential steps for performing a measure are:

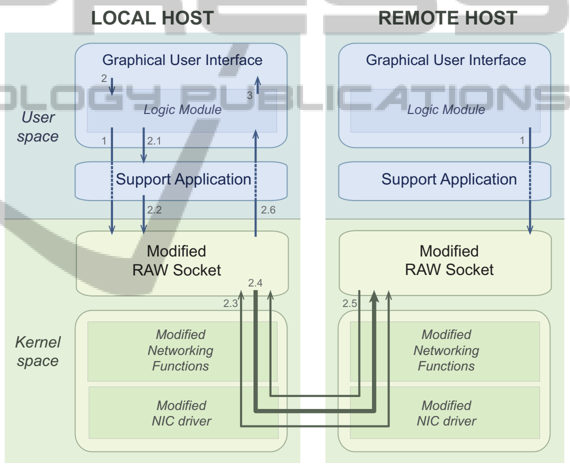

Figure 1: Sequential steps of the measurement procedure.

1. When the user starts the GUI application, it executes an instance of the Support Application which opens a Modified Raw Socket. This step takes place both in the local and the remote hosts.
2. When a capacity estimation is requested by the user, the Logic Module performs the different individual measures required for estimating that capacity. Therefore, it executes the process described in the Section 2.1. Each one of the individual measures are obtained as follow:

   2.1. The Logic Module sends the information of the individual measurement experiment to the Support Application.

   2.2. The Support Application module informs the Modified Raw Socket about the individual measurement taking place.

   2.3. Then, the Modified Raw Socket in the local host establish a connection with its homologue in the remote host. In this connection process, the remote host is informed about the number of packets in the individual measurement. Therefore, it knows when the individual experiment is finished.

2.4. If the connection has been successfully established, the local Modified Raw Socket starts sending the packets of the individual measurement to the remote host. In the top-half (Modified Networking Functions) of the remote host, the arrival-time of every packet of the experiment is measured and introduced in their payload. In the Modified Raw Socket of the remote host these times are extracted and stored.

2.5. After all the packets have been received, these times are sent back to the Modified Raw Socket in the local host using the socket communication.

2.6. The times are then sent to the Support Application using the natural socket raw process in Linux. Then, the Support Application estimate the capacity of the individual measure. This value is finally sent to the Logic Module.

3. After obtaining all the necessary individual measures, the logic module calculates the final estimation of the link-capacity requested by the user and the GUI displayes it.

5 RESULTS

For testing the tool, a test-bed consisting in a three link path connecting two hosts was implemented. The first and the third links had a $1Gbps$ capacity, whereas the second had $100Mbps$ capacity.

Using the developed tool and the timestamping system, the capacity of each one of the three links was measured. These results were compared with the results obtained measuring at socket level (i.e., after the bottom-half is executed) and without using any of the modifications explained before excepting the ones for disabling NAPI.

For obtaining the results shown in the sections below, each one of the interarrival-time measures were repeated consecutively fifty times in order to obtain a landscape picture of the behavior of both measure systems.

5.1 Measuring the First Link

For measuring the first link capacity, the second procedure described in the Section 2.1 is used. Thus, the first thing to do is to measure the end-to-end capacity of the path using a packet-pair to set r. Two packets of size $s(p) = 1500B$ were used for the packet-pair measure; therefore it was intended to measure an interarrival-time of $\Delta t = \frac{1500B \cdot 8\frac{b}{B}}{100 \cdot 10^6 \frac{b}{s}} = 120\mu s$.

The result of this experiment are shown in the first graph of Figure 2, where can be seen that using the improved timestamping system the measure is near $100Mbps$ in every repetition of the experiment. However, the traditional receiving scheme causes a very inaccurate and oscillatory measure which always overestimate the capacity.

Apart from that overestimation, the estimations obtained using interarrival-times measured at socket level show more variability that those obtained from times measured using the new timestamping system. This is caused by the variable time spent for executing the bottom halves and for switching context, and for the variable arrival-times caused by the NIC interrupt moderation system.

Once it is known that the capacity of the whole path is around $100Mbps$, the next step is to use a cartouche with $r \geqslant \frac{1000Mbps}{100Mbps} = 10$ for measuring the first link (step *2* in the second procedure of Section 2.1). The padding-packets will leave the path after the first link; this is done by setting their TTL to one.

In this manner, packets of size $s(p) = 1500B$ and a $r = 15$ were used to measure this capacity; resulting in the necessity of measuring at the remote host a time of $\Delta t = \frac{1500B \cdot (15+1) \cdot 8\frac{b}{B}}{1000 \cdot 10^6 \frac{b}{s}} = 192\mu s$.

As can be seen in the second graph of Figure 2, the values obtained using the developed tool are very accurate; whereas the estimations obtained measuring at socket level are worse than the obtained in the first measure, despite both times are quite comparable.

5.2 Measuring the Second Link

For measuring the second link, the first procedure of the Section 2.1 is used. In order to do that, the first thing to do is to measure the prefix composed of the first link. This is already been done in the Section 5.1, where it is shown to have a value of $1Gbps$.

Once this has been done, it is necessary to measure the capacity of the path composed of the first two links. This was done by using the padding-packets techniques with $r = 15$ and TTLs of two. This implies that it is necessary to measure at the remote host the time of $\Delta t = \frac{1500B \cdot (15+1) \cdot 8\frac{b}{B}}{100 \cdot 10^6 \frac{b}{s}} = 1920\mu s$. Note this time is much bigger than the time in the last two measures; therefore it would be expected that the measures in this case had less oscillations.

The results of this measures are shown in the third

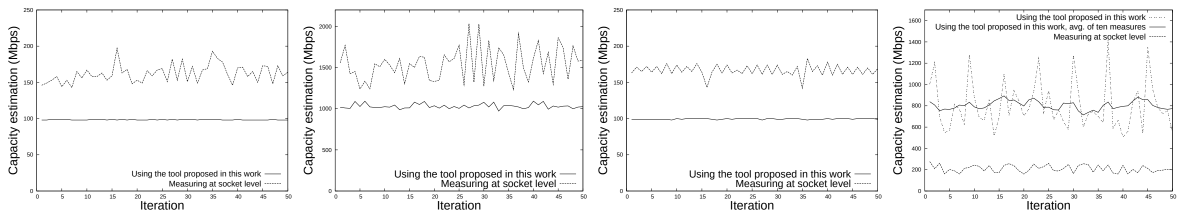

Figure 2: From left to right: capacity estimation of the whole path, capacity estimation of the first link, capacity estimation of the first two links, and capacity estimation of the last link.

graph of Figure 2. As can be seen, using the developments proposed in this work the results are more accurate. However, measuring at socket level causes the usual overestimation for the reasons already explained. As it was expected, the capacity estimations using the times measured at socket level has less oscillations than in the last two cases.

Like this capacity (about 100$Mbps$) is smaller than the capacity of the prefix composed of the first link (around 1$Gbps$), the first condition in step *2* of the first procedure in Section 2.1 is fulfilled and the final capacity estimation is the obtained in the second measure (100$Mbps$).

5.3 Measuring the Third Link

For measuring the last link, the first thing to do is to measure the capacity of the prefix composed by the first two links and compare it with the capacity of the whole path. In this case, the two measures has already been done, and they have been shown to have the same value.

Because of this, it is necessary to use the variable packet-size technique (step *3* of the procedure of Section 2.1), despite it requires measuring much smaller times. In this case, two packets with sizes $s(p) = 1500B$ and $s(m) = 100B$ were used. In this manner, it is necessary to measure at remote host a time of $\Delta t = \frac{100B \cdot 8\frac{b}{B}}{1000 \cdot 10^6 \frac{b}{s}} = 0.8\mu s$ which is a very small time. Due to its low value it would be expected less accurate and more variable measures.

The capacity estimations obtained from this last measure are shown in the fourth graph of the Figure 2. As shown in this figure, in this case the small time to measure cause a big variance and less accuracy in the capacity estimations obtained using the proposed method.

A simple method for removing the oscillation in the capacity estimation obtained using the proposed tool is to compute the average of a certain number of measures. In 2 the average of ten consecutive values of the capacity estimation is shown. Taking this average as final estimation, the last link-capacity is around 800$Mbps$. This implies an error of 20%. However, despite the lack of accuracy, the proposed system makes this kind of measures possible.

ACKNOWLEDGEMENTS

This research has been supported by project grant TEC2010-21405-C02-02/TCM (CALM) and is also promoted by the Aid Programme for Groups of Excellence run by Fundacion Seneca, an organ of the Murcia Region Science and Technology Agency (Regional Science and Technology Plan 2007/2010). Pedro J. Piñero-Escuer also thanks Fundacion Seneca for a Seneca Program FPI pre-doctoral fellowship (Exp. 16503/FPI/10). David Montoro-Mouzo also thanks the Fundacion Seneca for a pre-doctoral fellowship associated to the project ”FORMA” (Exp. 17541/BSCF/11).

REFERENCES

Benvenuti, C. (2005). *Understanding Linux Network Internals*. O’Reilly Media.

Bovet, D. and Cesati, M. (2005). *Understanding the Linux Kernel*, chapter 2. O’Reilly Media, 3rd edition.

Harfoush, K., Bestavros, A., and Byers, J. (2003). Measuring bottleneck bandwidth of targeted path segments. In *Proceedings of IEEE INFOCOM*.

Jacobson, V. and Karels, M. (1988). Congestion avoidance and control. In *Proceedings of SIGCOMM 88*.

Love, R. (2004). *Linux Kernel Development*. Sams Publishing.

Prasad, R. S., Murray, M., Dovrolis, C., and Claffy, K. (2003). Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE Network*, 17:27–35.